RSD-TR-28-86

High-Level Design of A Spray Finishing Robot Controller

by

Theordore A. Nesse

Kang G. Shin

The DeVilbiss Company
837 Ariport Blvd.
Ann Arbor, Michigan 48104

Department of Electrical Engineering
and Computer Science
The University of Michigan
Ann Arbor, MI 48109

December 1986

CENTER FOR RESEARCH ON INTEGRATED MANUFACTURING

Robot Systems Division

COLLEGE OF ENGINEERING

THE UNIVERSITY OF MICHIGAN

ANN ARBOR, MICHIGAN 48109-1109

TABLE OF CONTENTS

1. INTRODUCTION *1*

2. MODULARITY *2*

3. FAULT-TOLERANCE CONSIDERATION *4*

4. ARCHITECTURE *6*

4.1 Locigal Architecture for Module Interconnection *6*

4.2 Hardware Architecture for Module Interconnection *8*

5. SPRAY FINISHING ROBOT CONTROLLER TASKS *9*

5.1 Processing Modules *11*

5.2 Inter-Module Communication Rate *21*

6. SUMMARY *22*

7. REFERENCE *23*

ABSTRACT

Modularity and fault-tolerance are essential to the design of a spray finishing robot controller (SFRC). Vertical and horizontal communication architectures are discussed, and a horizontal, bus-connected architecture is proposed as the best architecture for the SFRC.

Tasks that must be supported by an SFRC are used to justify modularization decisions. These tasks are then partitioned and mapped into processing modules, basing module boundary decisions on criteria of system cost, inter-module communication needs, task processing requirements and module complexity.

The work reported here is supported in part by the NSF under Grant No. ECS-8409938 and the US AFOSR under Contract No. F33615-85-C-5105. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors do not necessarily reflect the view of the funding agencies.

1. INTRODUCTION

Most contemporary spray finishing robot controllers (SFRC's) do not take advantage of available hardware and software technologies that could lower the cost but increase the functionality of a controller. In some cases, the functionality of the controller appears to be determined by the capabilities of the board set that has been selected, rather than by the control needs of a spray finishing robot. As robot users become more sophisticated and demand higher performance, increasingly cost-effective robot controller designs will be required of robot manufacturers.

The following three limitations on the required functionality can be observed in most popular SFRC's [1] - [3]. First, constraints on local memory for program path storage and limited path processing capability require that a host computer be provided for support of robot consoles. Secondly, executable memory limitations constrain the functionality of the controller. As a result, special options such as a vision system interface are difficult to add to the system. Processing speed limitations in the servo system of this controller prevent the implementation of sophisticated servo algorithms as well. Thirdly, adding modules to the system is complicated by low bandwidth, proprietary protocol channels that are used to interconnect modules.

In these popular controllers, there is no emphasis on fault-tolerance, as none of the designs exhibit a coordinated approach to controlling hardware or software failures. While power-up self tests and some on-line diagnostic features are provided, they provide little help to an operator when failures occur. As these systems become more sophisticated and are used in larger numbers, fault-tolerance will become increasingly important.

This paper presents an SFRC architecture that specifically addresses the limitations mentioned above. Modularity is used as the guiding principle for the design, and is con-

sidered in detail in Section 2. Section 3 discusses fault-tolerance aspects that are appropriate for robot controllers, and an economical approach for providing a spare system bus is proposed. The SFRC architecture is presented in Section 4. First in Section 4.1, the logical architecture, as defined by inter-module communication paths, is discussed. Then hardware architectures that can support the selected logical architecture are discussed in Section 4.2.

As the hardware design for an SFRC must be directed toward providing effective support for the particular needs of spray finishing robots, it is necessary to identify the tasks that must be performed by that controller. Based on these tasks, processing modules required for the controller are identified in Section 5, and support for the module boundary decisions is provided. The functions of the modules are described in some detail, including identification of some useful features lacking in contemporary designs. The paper concludes with Section 6.

2. MODULARITY

Modular design has become standard practice in many disciplines, including robot control design. While the benefits of modular design may seem obvious, it is useful to examine some of them here to ensure that the expected benefits will be realized in this application. Note that the benefits of modular design apply to both the hardware and the software design of a robot controller.

The major benefits of modular design (as opposed to a highly integrated design) include improved maintainability, reduced implementation effort and reduced system complexity. A modular system is more readily maintained since personnel can restrict their system re-learning to the module in need of modification. Well-defined interfaces with other modules serve to limit the scope of the analysis required to design an effective

modification.

The benefit of reduced implementation effort is seen most clearly on systems where a team approach is required for the design work. Once the module boundaries are drawn and interfaces are defined, independent personnel can work efficiently on their own areas of the design with minimal impact on the other parallel design efforts.

The reduction in implementation effort is closely related to the reduced complexity that is evident in a modular design. While the overall system may be larger in a modular implementation (more connectors and drivers for hardware, communications and function calls for software), the resulting system is more comprehensible as it can be analyzed module by module.

System reliability can often be enhanced by the use of modular design techniques. In a system where modules are loosely-coupled, the failure of one module does not affect the operation of other modules. If a failed module is non-critical, system operation may continue. Testing to build confidence in a system is also eased by modular design techniques. In some cases, modules can be tested before they are integrated into the overall system, where errors may be more difficult to locate.

Given that modularity is a desirable characteristic in a design, determining the size of the modules for a given design can significantly affect the degree to which the benefits of modularity are realized. A system with many small modules may become burdened by a high level of inter-module communication, while a system with a few modules may lose the benefits noted above. Often, the application for the system may indicate reasonable boundaries for the modules. In a robot controller, for example, the servo system is clearly distinct from other controller functions and the interface with this module is straightforward to design. It is therefore desirable to form a module to support the

servo control tasks. In other cases, deciding what is to be included in a module requires careful evaluation of numerous conflicting factors.

3. FAULT-TOLERANCE CONSIDERATIONS

Modularity often forms a basis for fault-tolerant approaches. For example, triple modular redundancy (TMR) is based on the simultaneous operation of three modules which provide output to a voting element [9, p. 68]. Should one module fail, the voter will detect that its output is different from the other two modules, and only the information with majority votes will be released. Though there are technical challenges to implementing TMR for a robot controller, the key reason that this approach has not been taken in existing designs is that TMR has not been demonstrated to be a cost-effective approach to improving system reliability.

As robot controllers are rather prone to failure due to their complexity and harsh operating environment, some approach to tolerating failures is essential. Standby sparing at the controller level is a typical approach that is used to achieve acceptable system failure rates [8]. In such a standby sparing system, one robot can serve as a spare for ten or more robots. However, standby sparing does not provide uninterrupted service when failures occur, but is less costly than the replication of hardware that is otherwise needed for a system that can provide uninterrupted service.

Though techniques such as TMR may not be useful for robot controllers at the present time, there is no justification for ignoring opportunities to enhance system fault-tolerance. There are numerous techniques available to improve the reliability of hardware and software components of robot controllers that can be implemented economically. One recent development that promises to provide significant benefit to systems designers is the availability of both parallel and serial system buses in

commercially available board level products.

System bus failures are catastrophic for a system with a single bus, as all communication between modules is lost. Unfortunately, this is a common failure, since any one of many potential single point failures can disable a bus. A second bus is desirable in a bus-based system to allow such failures to be tolerated, but the typical hardware and software cost to implement such a dual bus system is difficult to justify for an industrial robot controller.

One solution to this problem is seen in Intel Corporation's Multibus II specification [7], where inter-module message passing can take place in a similar fashion on either the main parallel system bus or on an inexpensive serial system bus. Many operations can be carried out over either bus, with little software overhead to support a physically dissimilar channel. By virtue of the two wire implementation, the inclusion of a serial bus incurs very little increase in hardware cost either. While this hardware cost benefit is realized in both the Intel Multibus II specification [7] and the VMEbus specification [5], the fact that Multibus II uses the same protocol for message communication over the serial or parallel bus also provides the additional benefit that the software overhead is small.

The way this pair of system buses can be used to improve the fault-tolerance of a robot controller is as follows: During normal, fault-free operation, the parallel system bus can be used for normal message traffic between modules. The serial bus can be idle, or can be used to monitor the operation of the system's modules. Should the parallel system bus be disabled by the failure of a non-critical module, the robot controller can be designed to suspend all non-critical inter-module message traffic, and continue operation using the serial bus. The failed module can then be replaced at a convenient time, when

the robot is not required for production.

4. ARCHITECTURE

4.1. Logical Architecture for Module Interconnection

Before considering the physical means for interconnecting modules in a robot controller, the logical architecture needs to be defined. The concept of vertical versus horizontal communications that is useful at the system level can also be applied to the controller level of a system design [8]. If the modules of a robot controller are considered to form a hierarchical tree structure, as shown in Figure 1, simple vertical communication is seen when a module communicates with only its child modules. A more general form of vertical communication is observed when a module can communicate with any of its descendant modules. Horizontal communications are those communications that occur among modules that are not vertically related. If the communications are restricted to the children of a common parent, simple horizontal communications are seen. Contemporary SFRC's primarily use vertical communication architectures. The Allen Bradley 8200 [3] and Graco OM-5000 [2] controllers employ simple, vertical communications, while the DeVilbiss TR-4500 [1] controller includes some horizontal communication in a mostly vertical architecture. It has been noted in [8] that vertical communication is efficient, provided that the number of child modules is small, the modules are not modified often, the parent module is reliable and the tasks in the modules are not computationally intensive. In SFRC's, the first condition is met. The second condition is difficult to assess, but when modifications are needed in existing vertically designed controllers, the process is labor intensive and fraught with peril of introducing insidious timing bugs. The reliability of the parent modules is not well characterized in these controllers, but can be assumed to be the same as the child modules. The condition that tasks not be

Figure 1. Tree structure for robot controller modules.

computationally intensive is true for some modules, but not true for others such as coordinate conversion modules. Thus, a vertical communication architecture may or may not be appropriate for these controllers.

For a number of reasons related to those noted above, a horizontal architecture has been selected for the controller design being discussed here. As increasing functionality is required of robot controllers, to meet rising customer expectations, the above noted conditions for the efficient use of a vertical architecture will be invalidated. The addition of features such as visual and tactile sensing will add computationally intensive modules to the system. A further consideration favoring the use of a horizontal architecture is based on fault-tolerance. As the number of modules increases in a system, it becomes increasingly likely that a carefully designed system can tolerate the failure of a module. If this is the case, it is undesirable to have a single central controller module that is critical to system operation [10].

4.2. Hardware Architecture for Module Interconnection

Having selected a horizontal, distributed architecture for the controller, it is now necessary to consider the physical means for interconnecting the modules. As a starting point, the similarities between a robot controller and a distributed system can be noted. In a robot controller, modules are connected by communication links, as in a distributed system. Robot controller modules may be physically separated, such as when an operator interface module is remote from the main processing hardware of the controller. Due to the local intelligence in each module, only a limited amount of communication between modules is required, a further similarity with distributed systems.

Given the similarities to a distributed system, it is important to explicitly consider alternative interconnection architectures that could be used to establish communication channels between modules in a robot controller, rather than selecting a bus on the basis of historical precedent. There is no fundamental reason that module interconnection in a robot controller could not use a ring or star architecture, instead of the typical bus technique.

A ring architecture is attractive as it promotes good modularization of a system by establishing a well-structured communication system. Node failure is readily diagnosed, and system operation can continue with a single failed node if that node is non-critical. Handling communications to or from neighboring modules can add significantly to the processing load for a particular module, however.

Adding a central module and implementing a star architecture avoids the problem of burdening modules with communication for their neighbors, but at the expense of adding a module for this purpose. Good modularization is supported by this technique, but the central module becomes critical to the functioning of the system. Note that a

star architecture can be logically implemented within a bus system by constraining all modules to communicate through a single central module.

A conventional bus architecture provides the most general interconnection scheme, but at a high cost. Bus architectures also permit poor modularization practices in a system by allowing high data rates between any two modules. Bus architectures are very flexible, however, and this probably accounts for the widespread commercial availability of hardware supporting numerous standard buses. Within a bus-connected system, a system designer can logically specify any useful interconnection architecture, and thus tailor standard hardware to meet the needs of a specific application. It is the commercial availability of highly flexible bus connected hardware that leads a system designer to select a bus as the basic interconnection strategy in a robot controller.

5. SPRAY FINISHING ROBOT CONTROLLER TASKS

Two different levels of functionality can be envisioned for a robot controller. Typically, a robot is considered as a stand-alone system, where the controller must have the necessary functionality to support all of the operations required of the robot. In many applications, however, it is desired to combine several robot controllers into an integrated system that performs a single process, such as applying a top coat (of paint) to an automobile body. In a system consisting of several integrated robot controllers, some tasks can be eliminated from the functions that each controller is required to support. For example, the operator interface function is usually performed at a higher level than the robot controller in an integrated multi-robot system.

While there is a clear trend toward the use of robots in integrated systems, there are several reasons that this design will be directed toward supporting all of the functionality required of a stand-alone robot controller. First and foremost, much of the

commercial market for finishing robots is for stand-alone systems. Second, if the design problems for a stand-alone system are solved in a manner that is sensitive to the needs of an integrated system, the controller required for an integrated configuration can be obtained by eliminating modules from the stand-alone configuration. This design, therefore, is directed toward providing the full functionality required to support a stand-alone robot system. Providing for simple and reliable interfacing with an integrated automation system is a key design consideration for this system, however.

The items below, then, are the tasks that a stand-alone robot controller must support:

Program Storage: A robot controller must be able to store data defining the paths that a manipulator must traverse. Sufficient storage should be provided so that frequent reloading of the main storage to accommodate normal production patterns is not required. The storage must be non-volatile, withstand plant floor conditions and provide for backup operations.

Program Playback: An obvious function of the controller is to allow playback of a selected path. Related to this function is the need to implement servo control of the robot. Given that it is desirable to store programs in Cartesian coordinate form to allow them to be conveniently modified, the controller must also support kinematics functions during program playback to allow conversions between Cartesian coordinate data and the corresponding joint position (robot coordinate) data form.

Synchronization: Two types of synchronization are generally required of spray finishing robots: the tracking of moving parts and coordination with other robots.

Operator Interface: In a stand-alone system, a robot controller must implement a

highly capable ("user-friendly") operator interface to allow control of robot functions and to indicate system status.

Path Programming: Each robot controller must be able to interact with an operator to create and modify programs. In addition, the controller must be able to accept programs that have been generated off-line by other systems. This implies a need for a communications system that is able to handle large volumes of data.

I/O Control: The robot controller must be able to interact with non-programmable (application) devices in the work cell, typically using discrete electrical output signals. Some of this I/O support, such as spray gun triggering, must be closely synchronized with a robot's path. Other I/O, such as safety system monitoring, may occur asynchronously to the path.

Interface with Programmable Systems: A critical function to allow a robot controller to be integrated efficiently with automation systems is the ability to interface with external programmable systems. Such an interface would allow control of robot functions and system status indication.

Fault Diagnosis: In the event of robot system failure, the controller should be able to aid service personnel in locating defective hardware that needs to be replaced. This implies a need to maintain operator interface and inter-module communication even when faults are present in the system.

5.1. Processing Modules

The above list shows the tasks that a robot controller must support. The modules required in a system to support those tasks must be identified. Numerous criteria were

applied to different possible configurations to select the one that is presented. Functional grouping was considered so that tasks are incorporated into modules according to the similar functions each performs. This minimizes the amount of information that must be transferred between modules for coordination. Limiting inter-module communication was sometimes considered independently of coordination considerations. Cost was also a key consideration in the modularity decisions in this design. As hardware costs increase with the number of modules, the size of the modules in this configuration tends to be somewhat larger than might be expected for a system with modularity determined based strictly on functionality.

Figure 2 shows the proposed architecture for the SFRC. Each block on the diagram indicates a module formed of one or two circuit boards and processors. Two system buses are shown connecting the modules: one high performance parallel bus and one serial bus.

Inter-Module Communication: Before discussing the modules in detail, some comments on the inter-module communication links shown in the figure is warranted. All critical inter-module communication that normally would take place over the parallel bus would be designed so that it could be switched to occur over the serial bus if the parallel bus fails. It would be desirable to organize inter-module communications to use a bus independent message passing protocol so that the software required to allow the serial bus to back up the parallel bus would be minimal.

An SCSI protocol bus [6] is specified to connect mass memory devices to the communication/storage module. As SCSI is widely supported by suppliers of mass memory sub-systems, many options are available for tailoring the controller's main path program memory to an application. Bubble memory, floppy disk, micro disk, hard disk

Figure 2. Robot controller modules.

and RAM disk sub-systems are currently available with an SCSI interface. Specialized communication processor sub-systems can also be connected to the system through this bus, as exemplified by the Manufacturing Automation Protocol (MAP) module shown in the figure.

Since operator interface panels are often some distance from the console and are connected with field wiring, RS-422 is used as the physical channel to make this connection. The protocol on this channel is not critical, as it does not interface with external equipment. Any convenient low-level ASCII protocol can be selected for this link.

A particularly desirable feature to be designed into this system is to allow the operating software for each module to be loaded over the system bus and stored in electrically erasable programmable read only memory (EEPROM) in each module. In this way, each module can start up as soon as power is turned on, and gain the reliability benefit of non-volatile memory for executable code. Updates to the software would be more convenient than the typical process of physically replacing programmable read only memory integrated circuits, though. Whether this is actually feasible or not would have to be determined during the detailed design of the system. EEPROM components are relatively high priced (.08 cents/bit), and require eight times the board space that ultra-violet erasable programmable read only memories (UVPROM) require.

Servo Module: The servo module communicates with the rest of the system over the dual system buses that are specified for the design. It is unlikely that all of the hardware needed for this module can be purchased as a standard product, as many robot manipulators employ resolvers for position sensing. Resolvers require specialized analog signal handling that is not commonly available in compact form. The module will be configurable to support from three to eight high performance servo channels by separating the processing hardware from the interface hardware. In addition, two low performance servo control channels will be incorporated, along with support for synchronous I/O operations over an RS-422 serial channel.

The processing card will incorporate the processor, memory, serial I/O and system bus interfaces required by this module. The task performed by this module requires a high performance processor to reduce the level of complexity for the software. Advanced servo techniques for stabilizing high speed manipulators, such as the implementation of digital notch filters, will also require significant processor resources. Though the memory

requirements for this module are quite modest, enough memory should be available so that diagnostic and off-line functions can be supported.

The support for the synchronous I/O is quite simple. During playback, each point received by the servo module will have information attached to it defining the state of the synchronous outputs. Whenever this information changes, a message will be dispatched over a serial channel to remote hardware that de-multiplexes the I/O for use by application hardware. During teach operation, messages received on the serial channel from the remote hardware will define the state of the synchronous I/O data that is attached to the points being taught. The control of spray gun triggering, paint fan control and electrostatic voltage control are examples of I/O operations that must be synchronized. Note that analog I/O is required as well as digital I/O.

To interface with a separate card that implements the hardware interface with the external servo hardware, a local bus will be used. The Multibus II iLBX or VMXbus would be well suited for this. Though the servo interface card could not be purchased as a standard product, the processor board that drives it is a likely candidate for the use of a standard design.

The servo interface card does not contain any processing power, but provides a site for the conversion and buffering hardware required to interface with external equipment. Several configurations of this board are required, as different designs are used to match the controller to various external servo systems, thus containing the system hardware changes required for alternate manipulators to one electronic assembly. The various boards should be designed to accommodate incremental encoders, absolute encoders, resolvers, tachometers and potentiometers as measuring system input. Output from the card is designed to drive servo valves directly, or drive the external amplifiers required

for electric servo systems.

A critical design characteristic of these cards is a configuration register system that allows the servo processor card to identify the particular card that is installed in a system, and is able to configure the servo interface card under software control. In this module, and all others, the use of programming switches and jumpers will be avoided as incorrectly set manual configuration devices are a frequent cause of field hardware failures. Self-test circuitry is required too, to allow the processor card to perform off-line testing of the servo interface card.

The system partitioning that leads to this module is particularly simple to justify, as noted earlier. All of the servo axes in a robot system must be closely coordinated, and therefore should be controlled in the same module. The computational needs of this task approaches the capacity of a single processor, and therefore additional tasks cannot be placed in this module.

Main Storage/Communications Module: This module will implement the main storage for the path programs, and also support communication with external programmable equipment; primarily for the purpose of transferring path programs. In the event that an integrated network interface is implemented, it would also interface through this module. Network operations affecting only main storage are handled directly by this module, but other commands and status operations will be forwarded to the Operator Interface Module for arbitration.

The main reason for including the communications support in this module is that the communication with the host computer for path data transfer results in a great deal of data to be handled by the storage module. Placing the communication support in the same module as the storage hardware allows this process to occur without adding traffic

on the main system bus. The storage module would also have a surplus of processing power, without the addition of the communications function to its workload. While the decision to combine these functions is somewhat inconsistent with good practice for modularizing a system to reduce complexity, it is assumed that the use of a standard operating system for this module will allow a very simple interface with the mass storage, leaving only straightforward work to implement communications between the storage module and the rest of the system. This combination provides for low system bus usage and efficient use of processing resources with an acceptable increase in module complexity.

Some communication protocols are processing intensive, and require specialized hardware for implementation. MAP is an example of one such protocol. Using MAP, the system architecture can accommodate the specialized processor without major changes by implementing the communications processor module as a device on the SCSI bus. A high bandwidth path is established into the main storage module in this way, without changes to the rest of the system.

The selection of the hardware used for main storage depends on several factors which may be application-dependent, such as the environment the controller must withstand, number and type of programs required and cost sensitivity of the customer. The SCSI interface and use of a standard operating system for this module will allow a wide variety of devices to be supported.

The worst case size for this main memory can be estimated for a typical application as follows.

Assumptions:

-10 hours of path execution required

-16 bits resolution per axis

- 8 axes

- 8 bits resolution for path velocity

-48 bits of miscellaneous information per position

-10% storage overhead for directory, pointers

-5 mm accuracy desired at 1000 mm/sec velocity

-50 mm radius as tightest curve allowed for 1000 mm/sec

-"average radius" for a path curve is 200 mm

To reproduce a 50 mm radius with the desired accuracy, the point (position) spacing must be less than 43.6 mm, as shown in Figure 3. At 1000 mm/sec, the peak number of points per second for the system to handle will be 23 points per second in order to reproduce the 50 mm radius. An average data rate can be calculated in a similar manner from the 200 mm radius. The required minimum point spacing for the average case is 88.9 mm, yielding an average required rate of 10 points per second.

Figure 3. Calculation of minimum point spacing.

The number of bytes per position can be calculated based on the assumptions as follows:

(16 bits per axis) × (8 axes) + (8 bits velocity information)

+ (48 bits misc. information) = 184 bits (total bits per position)

(184 bits/pos.) × (1.1 overhead) = 202 bits per position

(202 bits/pos.) / (8 bits/byte) = 25 bytes per position.

This can be combined to give the desired memory size for the robot:

(10 hrs. per robot) × (3600 secs. per hr.) × (10 points per sec.) × (25 bytes per point)

= 9 megabytes per robot.

The estimate is highly sensitive to the paths that are executed, as straight paths dramatically reduce the volume of data required. The memory size required can change by an order of magnitude depending on the installation, but the estimate is consistent with observations of currently operational finishing robots containing one megabyte of local memory. In many cases, these controllers are connected to a host computer to provide additional memory.

Math Module: A module is required in the system to perform kinematics, linear interpolation and path velocity control. In a robot that is taught by leading the manipulator, forward kinematics must be performed to calculate the Cartesian data that is to be stored. During playback, data must be converted to robot coordinates (joint positions) for servo control. Like the servo module, this module will need a high performance processor. Some kinematics algorithms require large memory areas for tables, and therefore this module should be capable of interfacing with up to one megabyte of memory.

Operator Interface Module: Operator interface, programmable systems interface and asynchronous I/O are the three types of I/O supported by this module. The operator interface function consists of interaction with an operator panel over a serial RS-422 communication link. Messages to the operator are translated from the internal system message format to a form intelligible to the operator. Likewise, commands from the operator are converted to the internal message format. Menu or graphic manipulation to communicate with the operator will be invisible to the rest of the system until a command is selected that requires service of a module other than the Operator Interface module.

The programmable systems interface implements essentially the same functionality as the operator interface, but the data format is appropriate to communication with an external computer. As the data rate on this channel is not significant (a few bytes per second on the average), RS-422 is appropriate for this channel as well.

Asynchronous I/O also occurs over serial communication channels, much like that used for the synchronous I/O controlled by the servo processor. This I/O is not subject to the need to be synchronized with the manipulator path. Monitoring the internal temperature of a control console is an example of an asynchronous monitoring task. As long as it occurs at least every few minutes, the exact timing is not critical.

Diagnostics/Safety Module: An independent, simple, low performance processor is included in the system for the purpose of monitoring the operation of the system and aiding in diagnosing failures. During system start-up, this module interrogates the rest of the modules in the system to check the results of the self-test performed by each. During system operation, this processor can interrogate other processors to verify their correct operation in a general sense. A hard failure of the servo module, for example, could

result in an unsafe condition. Upon detecting loss of control by the servo processor, this module will initiate a controlled system shutdown.

In the event of a system failure, this module will initiate resident diagnostic routines to identify the failed hardware. While this module normally interfaces with the operator through the Operator Interface module, a failure in that module will disable the operator panel. For this reason, the Diagnostics/Safety module implements a simple interface for a standard ASCII terminal, to allow communication with the diagnostics system if the Operator Interface module fails.

5.2. Inter-Module Communication Rate

Having selected a configuration for the controller and considered the processing load for each module, it is necessary to ensure that the inter-module communication can be supported efficiently. It is important to note that each processing module is self-contained in this design. No processor fetches instructions using the system bus.

Figure 4 shows the system data flow when program playback is initiated by the operator. The messages generated are as follows:

- Operator Int. to Storage/Comm.: start playback - negligible data rate
- Storage/Comm. to Math: data to be converted - 250 bytes/sec.
- Math to Servo: actuator positions - 2500 bytes/sec.

Program teach operations follow a similar pattern:

- I/O to Storage/Comm.: prepare for teach - negligible data rate
- Storage/Comm. to Servo: start teach - negligible data rate
- Servo to Math: points to compress/convert - 2500 bytes/sec.

- Math to Storage/Comm.: points to store - 250 bytes/sec.
- I/O to Servo: halt teach - negligible data rate

As these data rates are orders of magnitude less than the capacity of even a typical synchronous serial bus (2 megabytes/sec), messages will pass essentially instantaneously. In addition, there are no inter-module communication needs that will result in the transfer of large blocks of data, further ensuring the timely transmission of inter-module messages.

6. SUMMARY

The high level design of an SFRC has been presented based on the tasks such a controller must perform. The tasks that must be supported by the controller are: Program Storage, Program Playback, Synchronization, Operator Interface, Path Program-

Figure 4. System data flow.

ming, I/O Control, Interface with Programmable Systems and Fault Diagnosis.

A bus architecture was selected as the physical means to interconnect the modules that were defined. A horizontal inter-module communications architecture was chosen, where no module has central authority in the system. Communication paths are established between any two modules which need to communicate, with control passed along the chain of modules thus established. It was shown that the inter-module message traffic will require orders of magnitude less bandwidth than current 32 bit buses can provide.

Based on functional partitioning, limiting inter-module communication and cost factors, the following modules were defined to form the system: Servo Module, Main Storage/Communications Module, Math Module, Operator Interface Module and Diagnostics/Safety Module.

This design provides a balance between numerous conflicting constraints. Safety and fault-tolerance were addressed in the design, but with consideration of the cost and complexity that could be tolerated for an industrial system. The design provides for simplicity, as only a minimum number of modules are specified. The characteristics of the module hardware are such that much of the hardware can be purchased off-the-shelf, reducing design costs.

REFERENCES

[1] The DeVilbiss Company, "TR-4500 Operator's Manual", *The DeVilbiss Company,* Toledo, Ohio, September 1985.

[2] Graco Robotics, "OM-5000 Operation Manual," *Graco Robotics Inc.,* Livonia, Michigan, 1983.

[3] Allen-Bradley Company, "Series 8200 Robot Control Specifications," Allen-Bradley Company, Highland Heights, Ohio, September 1983.

[4] "Robots man the paint booth at GM-Orion", *Robotics Today,* pp. 52-53, April 1985.

[5] *VMEbus Specification Manual,* Motorola Inc., Pub. no. M68KVMEB(D1), October 1981.

[6] Computer and Business Equipment Manufacturer's Association, "Draft proposed American National Standard for Information Systems - Small Computer System Interface", Standard No. X3. 131-198x Revision 17-B, December 13, 1985.

[7] *Multibus II Bus Architecture Specification Handbook,* Intel Corporation, Order Number 146077-B, 1983.

[8] K. G. Shin, "Intertask communications in an integrated multi-robot system", Center for Research on Integrated Manufacturing, The University of Michigan, RSD-TR-4-85. Also to appear in *IEEE J. Robotics and Automation.*

[9] T. Anderson and P. A. Lee, *Fault tolerance -- Principles and Practice,* Prentice Hall, London, 1981.

[10] S. Miyamoto, *et. al.*, "FMPA: A Fault-Tolerant Multi-Microprocessor System Based on Autonomous Decentralization Concept", *Digest of Papers,* FTCS-13, pp. 4-9, 1983.